REPORT DOCUMENTATION PAGE

AFRL-SR-BL-TR-00-

0619

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		1 June 1996 - 30 November 1998

4. TITLE AND SUBTITLE
A Formal Approach for the Design of Real-Time Distributed Systems

5. FUNDING NUMBERS
F49620-96-1-0221

6. AUTHOR(S)
Michael Evangelist

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Florida International University
Sponsored Research Office
University Park- PC 539
Miami, FL 33199

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR
801 North Randolph Street, Room 732
Arlington, VA 22203-1977

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-96-1-0221

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Real-time distributed systems are the backbone of the US Air Force computational infrastructure. The inherent complexity of these systems and their mission-critical nature present a difficult engineering challenge at every point in the lifecycle. This research is to lay a foundation for a systematic engineering methodology. In particular, our effort consists of two inter-related aspects. The first is to develop a scalable and flexible model and techniques for representing and analyzing distributed real-time architectures; and the second is to extend our previous work in automatic (correctness-preserving) transformations. The first part will proceed by integrating aspects of Petri net theory with a generalized object model to obtain a coherent formal approach to architectural representation. The second attacks the problems of scalability and excessive manual labor in the use of formal methods. In this final report, we summarizc the results and accomplishments of our research based on the above two aspects of this effort.

14. SUBJECT TERMS	15. NUMBER OF PAGES
	7
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

FINAL TECHNICAL AND INVENTION REPORT

AFOSR Grant No.: F49620-96-1-0221
Project Title: A Formal Approach for the Design of Real-Time Distributed Systems
Principal Investigators: PI: Michael Evangelist, Co-PIs: Yi Deng, Paul C. Attie
Organization: School of Computer Science, Florida International University

1. Overview

Real-time distributed systems are the backbone of the US Air Force computational infrastructure. The inherent complexity of these systems and their mission-critical nature present a difficult engineering challenge at every point in the lifecycle. This research is to lay a foundation for a systematic engineering methodology. In particular, our effort consists of two inter-related aspects. The first is to develop a scalable and flexible model and techniques for representing and analyzing distributed real-time architectures; and the second is to extend our previous work in automatic (correctness-preserving) transformations. The first part will proceed by integrating aspects of Petri net theory with a generalized object model to obtain a coherent formal approach to architectural representation. The second attacks the problems of scalability and excessive manual labor in the use of formal methods. In this final report, we summarize the results and accomplishments of our research based on the above two aspects of this effort.

2. Architectural Modeling and Analysis of Real-Time Systems

2.1. Problem Statement and Objective

Real-time systems are difficult to design because of several interlocking factors: they are often mission-critical in nature; they are complex and distributed reactive systems; and their behavior is time-dependent. To ensure the dependability of the system, rigorous formal design methods are needed. For a formal technique to be applicable to real world complex problems, however, an engineering methodology for applying the formal techniques must be developed. Our research in this project investigates open problems in the architectural representation, modeling, and design of distributed real-time systems, and is built upon our previous research results and technology developed under the sponsorship from NSF and USAF Rome Laboratory.

In particular, our research has three objectives. The first is to develop an integrated and incremental process of architecture modeling and analysis driven by system requirements or constraints. The second is to systematically ensure traceability between system requirements and design decisions, and to verify the conformance of the design to the requirements. The third is to effectively manage complexity in modeling and analysis, which implies that the model itself must be highly adaptive and changeable.

2.2. Major Accomplishments and New Findings

2.2.1. A Software Architectural Model

We have developed a Software Architectural Model (SAM) as the basis for modeling, designing and analyzing the architectural of real-time distributed systems. SAM provides a multiple leveled model and notation for describing different aspects of architecture level design such as structure, behavior and constraints. Its specification model can be characterized from several dimensions:

(1) Structurally, software architecture is specified as multi-layered compositions of components and connectors, which can be refined and analyzed individually.
(2) In SAM, the construction and refinement of architectural model is driven by system requirements (specified as architectural constraints) and their propagation. At each design level, SAM specifies not only the (operational) composition of system components, but also the (descriptive) constraints that the components and their composition must satisfy. Refinement goes in lockstep with the propagation of the constraints. SAM provides certain integrity rules (e.g. structural integrity, constraint consistency and refinement consistency) to assure design consistency. (More such rules are under development.)
(3) Notation-wise, SAM is integrative, which employs both model-oriented formalism (Petri nets) and property-oriented formalism (temporal logic). Software architecture is specified by a set-theoretical recursive description, where Petri nets are used to describe components and connectors, and temporal logic to specify architectural constraints. These two complementary notations are seamlessly integrated under the SAM framework. We have successfully applied SAM for the modeling and analysis of command & control systems and flexible manufacturing systems.

2.2.2. Timing Analysis of Real-Time System Architectures

To real time systems, timing constraints on end-to-end delay are the most important system-wide constraints established on the systems' external inputs and outputs. To ensure these original constraints are satisfied, each of constituent components will be subject to a set of derived intermediate component constraints. Since the original system-wide constraints allow many possibilities for the intermediate constraints, and designers derive them based on their understanding on the systems, an important issue is how to guarantee the consistency between system-wide constraints and intermediate component constraints.

Based on the above features of the SAM framework, we have developed a constraint-driven refinement and verification method to verify the consistency and conformance of system design against real-time constraints. Our analysis techniques combines refinement and analysis into an incremental process. Analysis is driven by constraint verification and carried out through the reachability analysis on Petri nets of temporal constraints with respect to reachable states.

First, based on SAM approach, we build a real time system's architecture model, formally specify its system-wide constraints by temporal formulae, and derive/define intermediate constraints for each component. Then, we derive a small and constant-sized time Petri net (TPN) model, which we call component requirement model, by translating the temporal formula representing the component constraint into its TPN form. Next, we plug the set of newly created TPNs into the real time system architecture model (also represented as a TPN), which results in a complete, i.e. executable net model. This TPN represents the model of the component constraints based on the system-wide architecture. In other words, if this TPN satisfies the system-wide constraints, then the component constraints are consistent to the system-wide constraints based on the real time system architecture. Finally, we verify if the constraint model satisfies the system-wide constraint patterns. A number of available techniques, e.g. reachability analysis, can be used for this verification.

There are three noteworthy features in our analysis technique:

(1) The analysis is compositional, in which a verified component is reduced to a much smaller Petri net based on the constraints imposed on the component. The architecture is consequently viewed as a composition of these smaller nets, and analyzed accordingly.
(2) The analysis is incremental across different design levels such that it is performed separately according to the different levels of abstraction.
(3) It is driven by satisfaction of architectural constraints, which is monitored during the construction of reachability tree, and terminated as soon as the goal is reached. It avoids the generation of a complete reachability tree and thus improves the efficiency.

2.2.3. Other Related Results

Related to our work on SAM, we have also developed scalable and compositional analysis techniques for performance evaluation of complex transportation networks and for throughput analysis of discrete event systems.

3. Correctness-Preserving Transformations

3.1. Problem Statement and Objective

We address the problem of refining formal specifications into code. Our approach is one of successive refinement using correctness-preserving transformations: we begin with an initial specification which we refine successively (by applying transformations) until a suitable low-level solution, from which code can be easily generated, is reached. The application of transformations is guided by the designer, and some of the transformations themselves may be implemented manually.

This work provides a means of verifying behavioral properties of complex distributed systems, and also attacks the problems of scalability and excessive manual labor in the use of formal methods. The objectives of this research are (1) to produce a useful set of correctness-preserving transformations, and (2) to contribute to a theoretical basis for designing and validating correctness-preserving transformations.

We also aim to maximize the degree of automation of the transformation process by ensuring that most transformations have applicability conditions that can be checked automatically.

3.2. Major Accomplishments and New Findings

Our major accomplishments are transformations for (1) producing a large concurrent program from many small concurrent rograms, i.e., a method of automatically composing many programs together, (2) producing an atomic read/write program from a high-atomicity program, and (3) producing a fault-tolerant program from a fault-intolerant one and a fault-tolerance specification.

We have also devised a general method for refining the liveness properties of distributed systems. This work is a contribution to objective (2) above. We discuss each of these accomplishments in turn.

3.2.1 Automatic Composition of Small Concurrent Programs

Methods for synthesizing large concurrent programs from specifications (expressed in, for example, temporal logic) have been proposed and investigated in the literature. An important advantage of automatic synthesis methods is that they obviate the need to manually compose a program and manually construct a proof of its correctness. One only has to formulate a precise problem specification; the synthesis method then mechanically constructs a correct solution. A serious drawback of these methods in practice, however, is that they suffer from the state explosion problem. To synthesize a concurrent system consisting of K sequential processes, each having N states in its local transition diagram, requires construction of the global product-machine having at least N^K global states in general. This exponential growth in K makes it infeasible to synthesize systems composed of more than 2 or 3 processes.

Our results show how to efficiently synthesize large concurrent systems. Our method does not explicitly construct the global state transition diagram of the program to be synthesized, and thereby avoids state explosion. Instead, it constructs a state transition diagram for each pair of component processes (of the program) that interact. This ''pair-program" embodies all possible interactions of the two processes. Our method proceeds in two steps. First, we construct a pair-program for every pair of ''connected" processes, and analyze these pair-programs for desired correctness properties. We then take the ''pair processes" of the pair-programs, and ''compose" them in a certain way to synthesize the large concurrent program. This composition is essentially an automatic correctness-preserving transformation: the correctness properties of the pair-programs are preserved in the large program, and the large program is obtained by syntactically combining the pair-programs.

3.2.2. Synthesis of an Atomic Read/Write Concurrent Program from a High-Atomicity Concurrent Program

Shared memory programs that use high-atomicity primitives (e.g., read and write a large number of shared variables in one atomic step) are easier to design and reason about than

programs that read or write only a single variable at a time (atomic read/write programs). Such high-atomicity programs can be regarded as being at an intermediate level between a specification and efficient code. We have devised a transformation from high-atomicity programs to atomic read/write programs. The method first refines the high-atomicity program in a straightforward way, which may introduce executions that violate the specification. It then generates the state-transition graph of the refined high-atomicity program, and eliminates all portions of this graph that violate the specification. Finally, it extracts a correct atomic read/write program from the refined graph.

3.2.3. Synthesis of a Fault-Tolerant Program from a Fault-Intolerant Program and a Fault-Tolerance Specification

We show how to synthesize a fault-tolerant concurrent program from a fault-intolerant program and a fault-tolerance specification. Our method generates the global-state transition diagram of the fault-intolerant program, adds to it all of the states reached by the occurrence of faults, and synthesizes the required recovery transitions from these fault-states. Thus, we transform a fault-intolerant program into a fault-tolerant one.

3.2.4. Refining Liveness Properties of Distributed Systems

Simulation relations have been used extensively to reason about concurrent program refinement: if every transition of an implementation has a corresponding transition at the specification level, then we say that the specification "simulates" the implementation. This implies that the possible behaviors of the implementation are a subset of the possible behaviors of the specification. We take this notion of behavior inclusion (technically speaking, trace inclusion) to be our notion of a correct refinement.

The great benefit of simulation relations is that they reduce reasoning about entire executions to reasoning about individual transitions only. Clearly, this is a significant reduction in the amount of formal labor required. To date however, this benefit has been realized for safety properties only; showing that liveness properties are preserved by the implementation still required reasoning over entire executions.

Despite the fact that reasoning about liveness is usually associated with reasoning over entire executions, variant functions, fairness etc., our method extends the simulation-relation method to liveness properties in such a way that it requires reasoning only over individual transitions. It thus presents a significant methodological advance over current methods.

4. Software Tools

None .

5. Person(s) Supported

Alan Spector, Esteban Jimenez, Chun Jin (graduate students)

6. Publications

1. P.C. Attie and E.A. Emerson, "Synthesis of concurrent programs for an atomic read/write model of computation," to appear in ACM Transactions on Programming Languages and Systems.
2. P.C. Attie, "Synthesis of large concurrent programs via pairwise composition," Proceedings of CONCUR'99: 10th International Conference on Concurrency Theory.
3. P.C. Attie, "Liveness-preserving simulation relations," Proceedings of the ACM Symposium on the Principles of Distributed Computing (PODC) 1999.
4. P.C. Attie and E.A. Emerson, "Synthesis of concurrent systems with many similar processes," ACM Transactions on Programming Languages and Systems, 20(1), January 1998, 51--115.
5. A. Arora, P.C. Attie, and E.A. Emerson, "Synthesis of fault-tolerant concurrent programs (extended abstract)," ACM Symposium on the Principles of Distributed Computing (PODC) 1998.
6. Y. Deng, J. Wang and R. Sinha, "Integrated Architectural Modeling of Real-Time Concurrent Systems with Application in FMS", Proceedings of International Conference on Software Engineering and Knowledge Engineering, June 1998, 34-43.
7. J. Wang, Y. Deng and X. He, "A formal architectural specification model for real-time systems design", Proceedings of the IASTED Conference on Software Engineering, October 1998, 11-14.
8. J. Wang and Y. Deng, "Component-level reduction rules for time Petri nets with application in C2 systems", Proceedings of 1998 IEEE International Conference on Systems, Man and Cybernetics, October 1998, 125-129.
9. Y. Deng and J. Wang, "Integrated Architectural Modeling and Analysis for High Assurance Command and Control System Design", Proceedings of 3rd International High Assurance Systems Engineering Symposium, November 1998, 270-278.
10. Y. Deng, J. Wang and R. Sinha, "Incremental Architectural Modeling and Verification of Real-Time Concurrent Systems", Proceedings of 2nd International Conference on Formal Engineering Methods, December 1998, 26-34.
11. Y. Deng and J. Wang, "Integrated architectural modeling and analysis for high-assurance command and control system design", Annals of Software Engineering, Vol. 7, 1999, 47-70.
12. J. Wang and Y. Deng, "Incremental modeling and verification of flexible manufacturing systems", Journal of Intelligent Manufacturing, Vol. 10, 1999, 485-502.
13. J. Wang, X. He and Y. Deng, "Introducing Software Architectural Specification and Analysis in SAM through an Example", Information and Software Technology - An International Journal, 41(7), 451-567, 1999

14. J. Wang, M. Zhou and Y. Deng, "Throughput analysis of discrete event systems based on stochastic Petri nets", International Journal of Intelligent Control and Systems, 3(3), 1999, 343-358.

15. J. Wang, C. Jin and Y. Deng, "Performance Analysis of Traffic Control System Based on Stochastic Timed Petri Net Models", Proceedings of 23rd International Conference on Computer Software and Applications Conference (COMPSAC), October 1999, 436-441.

16. J. Wang, C. Jin and Y. Deng, "Performance Evaluation of Traffic Control Networks via Stochastic Time Petri Nets", Proceedings of IEEE International Conference on Engineering of Complex Computer Systems, Las Vegas, Nevada, October 1999, 77-85.

17. J. Wang, G. Xu and Y. Deng, "Reachability analysis of real-time systems using time Petri nets", IEEE Transactions on Systems, Man and Cybernetics, in press.